

# Коректност на рекурсивни алгоритми

Тодор Дуков

## Няколко прости примера за рекурсивни алгоритми

Нека разгледаме следният алгоритъм:

```
1 int maximum(int *arr, int n)
2 {
3     if (n == 0)
4         return -INF;
5     else
6         return max(arr[n - 1], maximum(arr, n - 1));
7 }
```

Очевидно при параметри масив `arr` с размер поне `n`, функцията `maximum(arr, n)` връща  $\max arr[0 \dots n - 1]$ . Ще докажем това с индукция по `n`:

- за `n = 0` имаме, че:

$$\text{maximum}(\text{arr}, n) = -\infty = \max[] = \max arr[0 \dots 0 - 1], \text{ където под } [] \text{ се има предвид празният масив}$$

- за `n + 1` имаме, че:

$$\text{maximum}(\text{arr}, n + 1) = \max(\text{arr}[n], \text{maximum}(\text{arr}, n)) \stackrel{(\text{ИП})}{=} \max(\text{arr}[n], \max arr[0 \dots n - 1]) = \max arr[0 \dots n]$$

Ясно е, че сложността на алгоритъма може да се опише със следното рекурентно уравнение:

$$T(n) = T(n - 1) + 1 \text{ // базата няма да я пишем}$$

Директно се вижда, че  $T(n) \asymp n$ .

Да видим един малко по-сложен пример – за бързо степенуване:

```
1 int exp(int base, int power)
2 {
3     if (power == 0)
4         return 1;
5
6     int small = exp(base, power / 2);
7
8     if (power % 2 == 1)
9         return small * small * base;
10    else
11        return small * small;
12 }
```

С индукция относно `power`, ще покажем, че  $\text{exp}(\text{base}, \text{power}) = \text{base}^{\text{power}}$  (с уговорката, че  $0^0 = 1$ ):

- $\text{exp}(\text{base}, 0) = 1 = \text{base}^0$
- $\text{exp}(\text{base}, 2n + 1) = \text{base} \cdot \text{exp}(\text{base}, n) \cdot \text{exp}(\text{base}, n) \stackrel{(\text{ИП})}{=} \text{base} \cdot \text{base}^n \cdot \text{base}^n = \text{base}^{2n+1}$
- $\text{exp}(\text{base}, 2n + 2) = \text{exp}(\text{base}, n + 1) \cdot \text{exp}(\text{base}, n + 1) \stackrel{(\text{ИП})}{=} \text{base}^{n+1} \cdot \text{base}^{n+1} = \text{base}^{2n+2}$

Сложността на този алгоритъм може да се опише със следното рекурентно уравнение:

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Със Master теоремата лесно се вижда, че:

$$T(n) \asymp \log(n)$$

# Трик за бързо пресмятане на членове на някои рекурентни редици

Нека вземем за пример редицата на Фибоначи:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n+2) = F(n+1) + F(n)$$

Човек може да забележи, че:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} F(n+1) \\ F(n) \end{pmatrix} = \begin{pmatrix} F(n+2) \\ F(n+1) \end{pmatrix}$$

След това с индукция лесно се показва, че:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} F(1) \\ F(0) \end{pmatrix} = \begin{pmatrix} F(n+1) \\ F(n) \end{pmatrix}$$

За да сметнем  $F(n)$ , можем да направим бързо степенуване на матрицата, аналогична на тази с числата:

```
1 void multiply(int A[2][2], int B[2][2]) // записва резултата в A
2 {
3     int res[2][2];
4     res[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0];
5     res[0][1] = A[0][0] * B[0][1] + A[0][1] * B[1][1];
6     res[1][0] = A[1][0] * B[0][0] + A[1][1] * B[1][0];
7     res[1][1] = A[1][0] * B[0][1] + A[1][1] * B[1][1];
8
9     for (int i = 0; i < 2; ++i)
10         for (int j = 0; j < 2; ++j)
11             A[i][j] = res[i][j];
12 }
13
14 void fib_matrix_exp(int result[2][2], int n)
15 {
16     if (n == 0)
17     {
18         result[0][0] = result[1][1] = 1;
19         result[0][1] = result[1][0] = 0;
20         return;
21     }
22
23     fib_matrix_exp(result, n / 2);
24     multiply(result, result);
25
26     if (n % 2 == 1)
27     {
28         int fib_matrix[2][2] = {{1, 1},
29                                   {1, 0}};
30         multiply(result, fib_matrix);
31     }
32 }
33
34 int fib(int n)
35 {
36     if (n <= 1)
37         return n;
38
39     int matrix[2][2];
40     fib_matrix_exp(matrix, n - 1);
41
42     return matrix[0][0];
43 }
```

Коректността и сложността на алгоритъма оставяме на читателя (напълно аналогично е на предният алгоритъм).

В общият случай ще имаме рекурентно уравнение от вида:

$$T(n+k+1) = a_k T(n+k) + \dots + a_1 T(n+1) + a_0 T(n), \text{ където } a_0, \dots, a_k, k \text{ са константи}$$

Тогава имаме следната зависимост:

$$\begin{pmatrix} a_k & a_{k-1} & a_{k-2} & \dots & a_0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} T(n+k) \\ T(n+k-1) \\ T(n+k-2) \\ \vdots \\ T(n) \end{pmatrix} = \begin{pmatrix} T(n+k+1) \\ T(n+k) \\ T(n+k-1) \\ \vdots \\ T(n+1) \end{pmatrix}$$

Отново с индукция лесно се показва, че:

$$\begin{pmatrix} a_k & a_{k-1} & a_{k-2} & \dots & a_0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} T(k) \\ T(k-1) \\ T(k-2) \\ \vdots \\ T(0) \end{pmatrix} = \begin{pmatrix} T(n+k) \\ T(n+k-1) \\ T(n+k-2) \\ \vdots \\ T(n) \end{pmatrix}$$

## Задачи

*Задача 1.* Да се напише алгоритъм `calculate(first_members, step, k, n)`, който приема два масива от цели числа `first_members` и `step` с размер `k+1`, естествено число `n`, и връща `n`-тия член на следната рекурентна редица:

$$\begin{aligned} T(0) &= \text{first\_members}[0] \\ T(1) &= \text{first\_members}[1] \\ &\vdots \\ T(k) &= \text{first\_members}[k] \\ T(n+k+1) &= \text{step}[k] \cdot T(n+k) + \dots + \text{step}[1] \cdot T(n+1) + \text{step}[0] \cdot T(n) \end{aligned}$$

След това да се докаже неговата коректност, и да се изследва сложността му по време.

*Задача 2.* Даден е следният алгоритъм:

```

1 void stooge_sort(int *arr, int l, int h)
2 {
3     if (l >= h)
4         return;
5
6     if (arr[l] > arr[h])
7     {
8         int temp = arr[l];
9         arr[l] = arr[h];
10        arr[h] = temp;
11    }
12
13    int t = (h - l + 1) / 3;
14
15    if (t >= 1)
16    {
17        stooge_sort(arr, l, h - t);
18        stooge_sort(arr, l + t, h);
19        stooge_sort(arr, l, h - t);
20    }
21 }
```

Да се докаже, че при подаден масив `arr` с размер `n`, функцията `stooge_sort(arr, 0, n-1)` ще сортира `arr`. След това да се направи анализ на сложността по време.